



UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA
INDUSTRIALE E DELL'INFORMAZIONE

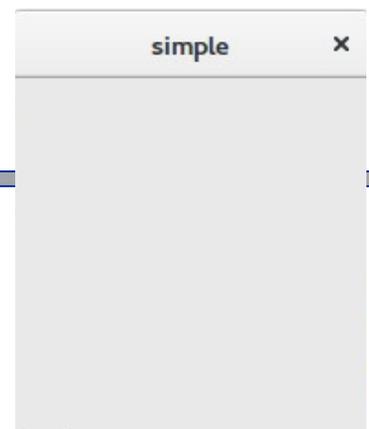
GTK+

Prof. Salvatore Venticinque

Prof. Massimiliano Rak



Hallo World



```
#include <gtk/gtk.h>
```

```
int main (int argc, char *argv[])
```

```
{
```

```
    GtkWidget *window;
```

```
    gtk_init(&argc, &argv);
```

```
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

```
    gtk_widget_show (window);
```

```
    gtk_main ();
```

```
    return 0;
```

```
}
```

Puntatore a Component

Creazione Finestra

Visualizzazione Finestra

Attesa Eventi



Compilazione

```
gcc -o esempio esempio.c `pkg-config --cflags --libs  
gtk+-3.0`
```



gtk_init()

- Prima di usare le librerie GTK+, occorre inizializzare l'applicazione, ovvero connetterla alla componente del sistema operativo di gestione finestre.
- E' possibile utilizzare ***gtk_init_check()*** per controllare se il servizio è disponibile e, in caso contrario far partire l'applicazione in modalità testuale.
- Nell'esempio precedente viene creato, visualizzato il componente grafico.



gtk_main()

- Come tutte le librerie per lo sviluppo di interfacce grafiche GTK+ utilizza un modello di programmazione ad eventi (event-driven).
- Quando l'utente non interagisce con l'interfaccia grafica, l'applicazione attende in un loop infinito.
- Quando avviene un evento (come il click del mouse) il sistema si sveglia e passa l'evento a uno o più componenti.
- Quando un componente riceve un evento, notifica uno o più segnali al programma.



Processo di sviluppo

- Disegnare la finestra
- Scegliere gli eventi
- Definire le azioni da svolgere in corrispondenza degli eventi
- Collegare le azioni agli eventi relativi ai componenti
- Visualizzare le componenti
- Attendere gli eventi



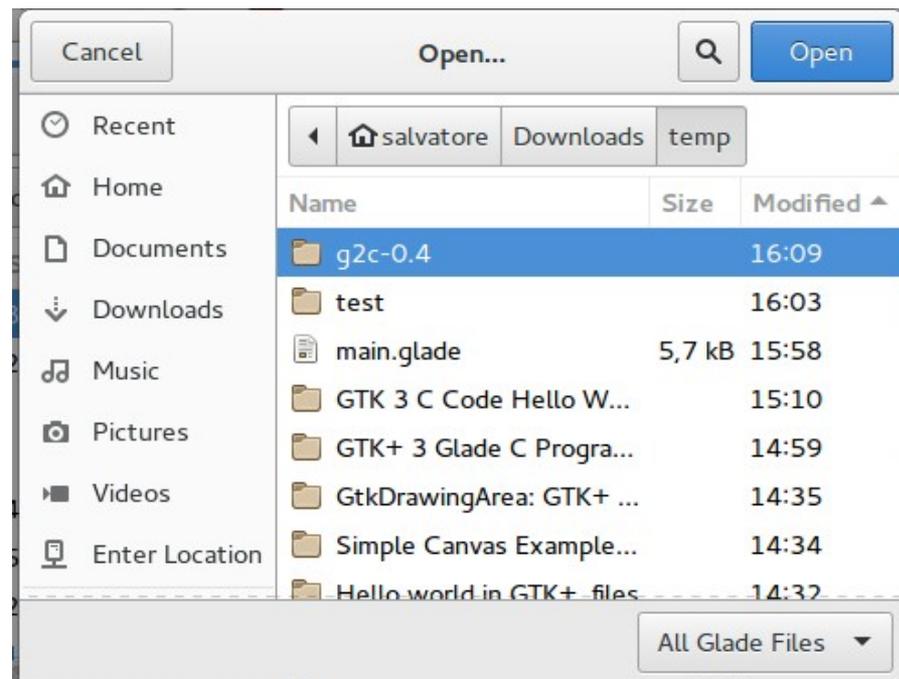
Disegnare una interfaccia grafica

- Dichiarare puntatori a componenti
- Creare i componenti e inizializzarli
 - Top Level
 - Containers
 - Display e control
- Comporre i componenti

Top Level Widgets

Top level

- **Application Window**
- **FileChooser**
- Application Chooser
- Message Dialog
- Color Chooser





Control and Display

- **Label:** Etichetta
- **Entry:** Casella di testo
- **Button**
- Radio
- Checkbox
- Toggle
- Textview

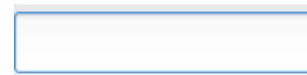
Creazione etichetta:



```
Widget * label = gtk_label_new ("Enter some text: ");
```

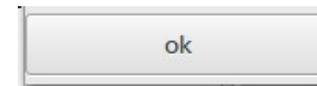
Creazione casella di testo:

```
Widget * entry = gtk_entry_new ();
```



```
gtk_entry_set_max_length (GTK_ENTRY (entry),0);
```

Creazione pulsante:



```
Widget * button = gtk_button_new_with_label ("ok");
```



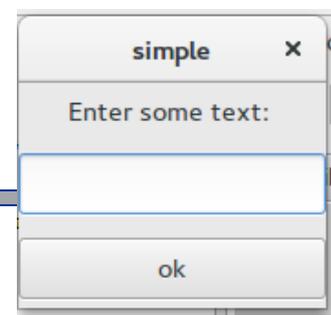
Definiscono il layout della finestra

Containers

- **Grid**
- **Box**
- Scrolled
- Menu Bar
- Toolbar



Esempio Vertical Box



// Creazione di un box verticale con 3 righe

```
vbox = gtk_box_new  
      (GTK_ORIENTATION_VERTICAL,3);
```

```
gtk_box_pack_start(GTK_BOX(vbox), label, TRUE, TRUE, 5);  
gtk_box_pack_end(GTK_BOX(button), entry, TRUE, TRUE, 5);  
gtk_box_pack_end(GTK_BOX(vbox), entry, TRUE, TRUE, 5);
```

Expand, fill, padding

Aggiunta di un layout a Griglia

```
/* Creazione container griglia */
```

```
grid = gtk_grid_new ();
```



```
/* Aggiunta del container alla finestra*/
```

```
gtk_container_add (GTK_CONTAINER (window), grid);
```

```
/* Aggiunta dei pulsanti al container*/
```

```
gtk_grid_attach (GTK_GRID (grid), button, 0, 0, 1, 1);
```

```
gtk_grid_attach (GTK_GRID (grid), button, 0, 1, 1, 1);
```

```
gtk_grid_attach (GTK_GRID (grid), button, 1, 0, 2, 1);
```



Funzione di libreria

```
Void gtk_grid_attach (GtkGrid *grid,  
                    GtkWidget *child,  
                    gint left,  
                    gint top,  
                    gint width,  
                    gint height);
```



Gestire gli eventi

- Quando un oggetto riceve un evento, notifica un segnale al programma.
- Per gestire l'evento occorre **connettere** un'azione al segnale con il metodo:
g_signal_connect()
- L'azione viene definita come funzione
- Una funzione connessa a un segnale è detta **"callback"**.
- Alla fine del callback, GTK+ ritorna ad eseguire il ciclo infinito nell'attesa di altri eventi.



Collegare azione a button

- collegare la funzione ***myButtonClicked*** all'evento ***click*** di ***button***

```
g_signal_connect(button, "clicked",  
    G_CALLBACK(myButtonClicked), NULL);
```

- collegare la chiusura del programma (***gtk_main_quit***) all'evento ***destroy*** di ***window***

```
g_signal_connect (window, "destroy",  
    G_CALLBACK(gtk_main_quit), NULL);
```



Codice Callback

```
void myButtonClicked()  
{  
    g_print("clicked\n");  
}
```



Passare parametri al callback

```
g_signal_connect(button, "clicked", G_CALLBACK(myButtonClicked),  
    entry);
```

```
void myButtonClicked2(GtkWidget *widget, GtkEntry *entry)  
{  
    GtkEntryBuffer * buffer=gtk_entry_get_buffer (entry);  
    g_print("clicked %s\n",gtk_entry_buffer_get_text(buffer));  
}
```

I parametri passati sono:

- Il puntatore all'oggetto che ha notificato il segnale
- Il puntatore a un qualsiasi altro oggetto GTK+



Strumenti visuali

- Nessuno realizza le Interfacce grafiche tutte programmando
- Tutti gli ambienti offrono strumenti visuali
- Occorre comunque scrivere i ***callbacks***



Glade

- Consente lo sviluppo visuale
- Consente di testare l'interfaccia
- Non esporta il sorgente, ma una descrizione xml dell'interfaccia
- Una funzione GTK+ legge dal file e crea dinamicamente il componente disegnato



Glade

Callback

The screenshot shows the Glade GUI designer interface for a file named 'main.glade'. The main workspace displays a window titled 'window1' containing a 'File' dialog box with a 'Filename' entry field and an 'Open' button. Below the dialog are four buttons. The right-hand pane shows a widget hierarchy tree with 'open_f' selected. Below the tree is the 'Signal' table for the selected widget.

Signal	Detail	Handler	User data	Swap	After
activate		<Type here>	<Click here>	<input type="checkbox"/>	<input type="checkbox"/>
clicked		open_f	<Click here>	<input type="checkbox"/>	<input type="checkbox"/>
enter		<Type here>	<Click here>	<input type="checkbox"/>	<input type="checkbox"/>
leave		<Type here>	<Click here>	<input type="checkbox"/>	<input type="checkbox"/>
pressed		<Type here>	<Click here>	<input type="checkbox"/>	<input type="checkbox"/>
released		<Type here>	<Click here>	<input type="checkbox"/>	<input type="checkbox"/>



Utilizzo file Glade

```
int main(int argc, char *argv[])  
{  
    GtkBuilder *builder, *window;  
    gtk_init(&argc, &argv);
```

```
    builder = gtk_builder_new();  
    gtk_builder_add_from_file (builder, "main.glade", NULL);  
    window = GTK_WIDGET(gtk_builder_get_object(builder, "window1"));  
    gtk_builder_connect_signals(builder, NULL);  
    g_object_unref(builder);
```

```
    gtk_widget_show(window);  
    gtk_main();  
    return 0;
```

```
}
```

```
void open_f()  
{  
  
    g_print("clicke\  
n");  
}
```



Compilazione

```
gcc -o esempio esempio.c `pkg-config --cflags --libs  
gtk+-3.0` -export-dynamic
```



Creare un'area per disegnare

- Creare l'area di disegno:

```
drawing_area = gtk_drawing_area_new ();
```

- Settare le dimensioni:

```
gtk_widget_set_size_request (drawing_area, 400, 400);
```

- Connettere il metodo di disegno:

```
g_signal_connect(G_OBJECT(drawing_area),"draw",  
G_CALLBACK(draw_callback),NULL);
```



Il metodo che disegna

```
gboolean draw_callback(GtkWidget *widget,cairo_t  
*cr,gpointer data)
```

```
{
```

```
    GdkRGBA color;
```

*X, Y, raggio, angolo1, a
angolo2*

```
    cairo_arc (cr,200, 200,200,0, 2 * G_PI);
```

```
    gdk_rgba_parse (&color, "#FFFFFF");
```

```
    gdk_cairo_set_source_rgba (cr, &color);
```

```
    cairo_fill (cr);void
```

*cairo_stroke (cr)
disegna solo la
circonferenza*

```
}
```

Per ridisegnare il widget da un callback:

```
gtk_widget_queue_draw (GtkWidget *widget);
```